In AAAI Spring Symposium on Extended Theories of Action: Formal Theory and Practical Applications, 1995

Representation and Evaluation of Plans with Loops

David E. Smith

Rockwell International
444 High St.
Palo Alto, California 94301
de2smith@rpal.rockwell.com

Mike Williamson

Department of Computer Science & Engineering
University of Washington
Seattle, Washington
mikew@cs.washington.edu

1 Introduction

Figure 1 shows a fragment of an assembly process plan for

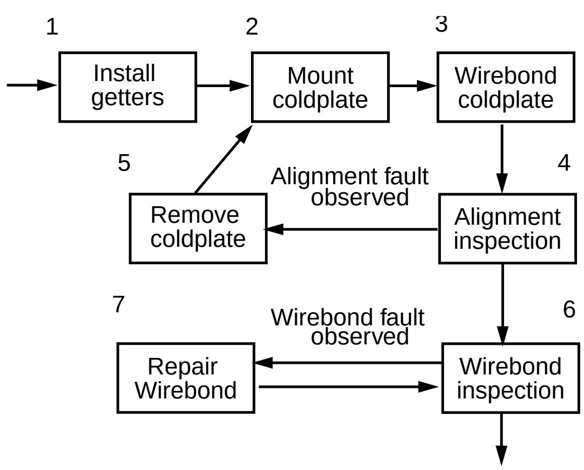

Figure 1: Assembly plan fragment for IR-FPA dewar assembly.

an infra-red focal plane array (IR-FPA) dewar assembly (cryogenic thermos bottle). There are several interesting things about this plan:

- Assembly steps such as 1, 2, and 3 can stochastically introduce faults into the device.
- Tests steps such as 4 and 6 can detect certain faults, but the tests are not perfect: they may fail to detect existing faults or report the presence of non-existing faults.
- Repair steps, such as 5 and 7, may stochastically repair existing faults and/or introduce new faults.
- The plan has conditional branches and loops.

This paper describes:

1. A formal representation for such plans; it allows probabilistic, information-gathering actions, parallel execution, contingent execution, and loops. (Most existing plan representations could not express loops.)
2. An aggregation technique for evaluating such plans. This technique uses Markov modelling to recursively evaluate and simplify small fragments of the schematically represented plan. We are thus able to avoid constructing a Markov model for the entire plan.

2 Representing Plans with Loops

We represent steps in an assembly process plan with a probabilistic, propositional action representation equivalent to that of C-BURIDAN [Draper94]. Each *step* has a set of *outcome distributions* that depend mutually exclusively and exhaustively on the state of the world when the action is executed. Each outcome distribution is a probability distribution over outcomes, where each outcome consists of:

- STRIPS-like lists of domain propositions that are added and deleted by the outcome;
- similar lists of *observable* propositions that are added and deleted by the outcome; and
- the cost associated with that outcome.

As an example, Figure 2 shows the representation of test step 6 and repair step 7:

The observable propositions represent statements not about the domain itself, but rather about which of the discernible classes of outcomes occurred when the action was executed (see [Draper94] for a discussion). A plan's control flow can only be contingent upon the value of observable propositions.

Previous plan representations consisting of simple or even partially ordered sequences of actions are insufficient for expressing plans with loops. Our representation is more complex; we will first give a rough overview, then a more precise characterization. Informally, a plan is a labeled graph (as in Figure 1), with nodes representing plan steps, and arcs representing control flow through the plan. Each of the steps in a plan can be partially or fully *enabled*. At each time increment, any fully enabled step may be executed, after which it will be disabled. Generally, execution of a

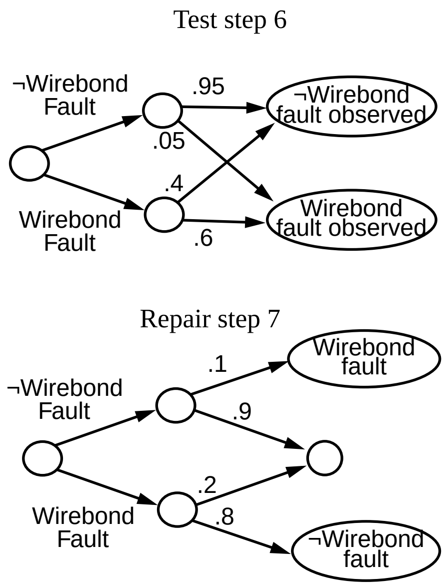

Figure 2: Representation of steps 6 and 7.

step will enable one or more following plan steps, perhaps conditionally on the observed outcome of the present or other earlier steps. A plan step may enable more than one following step, allowing execution to *fork*. Multiple threads of execution are *joined* through the use of partial enablement. A step may be partially enabled by each of two (or more) predecessors, and only when both of the predecessors have enabled the step will it be fully enabled.

Formally, a plan is a collection of *plan elements*, each a three-tuple $<\sigma, \pi, \Psi>$, where:

- σ is a plan step;
- π, the *enablement condition*, is a non-empty set of unique enablement symbols; and
- Ψ, a *conditional enablement descriptor*, is a set of pairs $<\kappa_i, \varepsilon_i>$ where κ_i is a formula of observable propositions, and ε_i is a (possibly empty) set of enablement symbols. (When there is only a single pair and κ = *true* we abbreviate with just the enablement descriptor ε.)

Associated with each plan is an *initial enablement* (a set of enablement symbols). During execution, the *current enablement*, χ, (also a set of enablement symbols) is maintained. When an element's enablement condition is a subset of the current enablement (i.e. $\pi \subseteq \chi$), that element may be executed:

1. the elements of π are removed from χ;
2. the associated step, σ, is executed;
3. each of the κ_i is evaluated in the context of the current state of the observable propositions. If the conditions hold, the elements of the associated ε_i are added to χ.

Execution terminates when no elements are fully enabled. Using this notation, the plan fragment in Figure 1 could be represented as:

```
<Install getters, e1, e2>
<Mount coldplate, e2, e3>
<Wirebond coldplate, e3, e4>
<Alignment inspection, e4,
        {<Alignment fault observed, e5>
         <¬Alignment fault observed, e6>}
<Remove coldplate, e5,e2>
<Wirebond inspection, e6,
        {<Wirebond fault observed, e7>,
         <¬Wirebond fault observed, e8>}
<Repair wirebond, e7, e6>
```

3 Plan Evaluation

For the plan fragment in Figure 1, each step has associated labor and materials costs. The entire plan can therefore be evaluated with respect to two attributes: its *expected cost*, and its *yield*, that is, the probability that the final product has no faults. [Kushmerick94] describes several techniques for evaluating probabilistic plans without loops, but these techniques do not easily extend to plans containing loops.

For plans with loops, one approach is to construct a Markov chain representing execution of the plan. Our plans give rise to finite Markov chains with transient nodes representing various states that arise during the execution of the plan, and absorbing nodes representing the various outcomes of the plan. Transitions from a state represent the various outcomes of executing one of the enabled actions in that state, and have an associated probability and cost. These Markov chains can be constructed automatically from the plan and step representations given above. A standard analysis technique for Markov chains (e.g. [Winston94]) will determine a probability distribution over the absorbing states and the expected number of times that each transient state is visited. This is precisely the information that we need to determine the plan's yield (i.e. the sum of the probabilities of absorbing states in which the goal is achieved) and it's expected cost (i.e. the sum of the expected number of times that each transition is taken times the cost of that transition).

In practice, however, this approach is infeasible, due to the extremely large size of the resulting Markov chain. States in the Markov chain must encode all feasible combinations of domain propositions, observable propositions, and current enablements. For a realistic model of IR-FPA assembly the plan has 107 elements, 57 domain propositions, and 23 observable propositions. This gives rise to a Markov chain with 24697 states, which takes about 58 minutes to construct and analyze. While this could be used for detailed analysis of a finished plan, it is far too slow for use in a planning or optimization algorithm where hundreds or even thousands of candidate plans must be considered.

4 Plan Space Aggregation

The basic problem with the Markov chain approach is that if there are *n* possible faults at a given point in the plan, there are $O(2^n)$ possible states in the Markov chain. However, a typical assembly action in the plan is unaffected by most of the possible faults. The Markov chain does not take advantage of this independence. We are therefore developing an alternative approach to plan evaluation that exploits this independence using the much more compact plan representation. The basic idea is to recursively:

1. Choose a small fragment of the plan.
2. Build and solve Markov chains for that fragment, ignoring all properties (faults) that steps in the fragment do not depend on or influence.
3. With the results of 2, build an aggregate contingent plan step representing the fragment.
4. Replace the fragment with the aggregate step.

For the example in Figure 1, steps 2, 3, 4, and 5 can be aggregated into the single step shown in Figure 3a, and

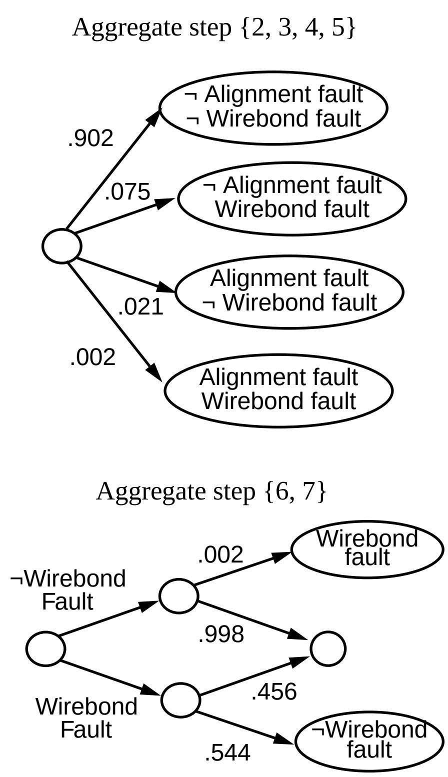

Figure 3: Aggregate steps for the step sets{2, 3, 4, 5} and {6, 7}.

steps 6 and 7 can be aggregated into the single step shown in Figure 3b.

Having done this, the entire plan fragment in Figure 1 can be aggregated into a single step with essentially the same structure (different numbers) as that in Figure 3a. The advantage of this approach is that the aggregation of {2, 3, 4, 5} and {6, 7} were done only once, without having to consider all possible combinations of earlier independent faults. In particular, the aggregation of {6, 7} did not have to consider alignment faults. In contrast, solution of the Markov chain (or Monte Carlo simulation for that matter) would solve these problems once for every possible combination of independent faults that might be present.

Fragment selection. One subtlety in the aggregation process has to do with fragment selection. A fragment is a subset of the elements in the plan, but not just any subset will work. To allow aggregation, a set of steps must have a single outside entry point. More precisely, if the plan is represented as a graph, all enablement arcs coming into the fragment's subgraph must come in to a single step. Otherwise it is not possible to produce a single aggregate step representing the behavior of the fragment. In the plan of Figure 1, the step sets {2, 3}, {2, 3, 4} and {2, 3, 4, 5} are all legal fragments because all outside enablements come in to step 2. Conversely {1, 2} would not be a legal fragment because both steps 1 and 2 have outside enablements.

The choice of fragment also has a significant impact on the efficiency of the entire process. Collapsing linear sequences of steps does some good, but the biggest payoff appears to come from aggregating loops into single steps. Thus the best approach appears to be to start from the inside and collapse small loops, working outward to surrounding loops. We are investigating a number of heuristic strategies for fragment selection.

Aggregation. A second subtlety in the aggregation process is that building an aggregated step may involve the solution of more than one Markov chain. The steps in a plan fragment can be conditional (like steps 6 and 7), so a different Markov chain must be built and solved for each possible combination of those contingencies that occur on entry to the fragment. In our example, steps 6 and 7 both depend on whether or not there is a wirebond fault. Furthermore, a wirebond fault can occur at step 4, prior to entry into the fragment. As a result, two Markov chains must be solved, one for the case where a wirebond fault is present on entry to the fragment, and the other where there is no wirebond fault on entry to the fragment. The solution of these two Markov chains map into the two branches in the aggregate model shown in Figure 3b.

In general, the relevant conditions for a fragment can be determined by taking the union of the conditions of all steps in the fragment. For the fragment {6, 7} the relevant condition is “wirebond fault”. Deciding whether or not these conditions can vary on entry to the fragment is more difficult – it requires a search backwards through the plan graph to see if any preceding steps can affect those conditions.[1]

Preliminary Results. Our preliminary experiments with aggregation have been limited to combining consecutive non-branching plan steps. For the realistic IR-FPA assembly plan this reduces the number of plan steps from 107 to 82 in 3.7 seconds. This reduces the size of the complete Markov model from 24,697 steps down to 17,825, and cuts the model building and solution time in half. We expect that aggregation of loops will provide much more significant savings.

In addition to analytical solution of Markov chains we have also considered using Monte Carlo simulation to evaluate plans directly in the plan representation. Although this is faster than analytical solution, it still appears to be too slow for use in planning or optimization. However, we could use Monte Carlo simulation (instead of Markov chain solution) to evaluate and simplify fragments in our aggregation technique. We have not yet investigated this alternative.

5 Conclusions

Analytical solution of Markov chains provides a sound basis for evaluating plans involving probabilistic, information-gathering actions, parallel execution, contingent execution, and loops. Unfortunately, the state space for the Markov model is huge for realistic assembly process plans. The problem is that Markov chains do not take advantage of the fact that most assembly steps are independent of most of the possible faults that can occur in previous steps. The Markov chain replicates state transitions over and over for many combinations of irrelevant faults.

To fix this problem we have introduced a plan space aggregation technique that isolates small, related fragments of the plan, and uses Markov chains to reduce these fragments to individual plan steps. We conjecture that this approach will allow exponential reduction in the solution time for assembly process plans, where the impact of faults is localized.

[1] Actually it is more difficult than this. A fault might be introduced by a step and later corrected through a perfect inspection and repair procedure. Thus, it is easy to determine an upper bound on the set of relevant, varying entry propositions, but finding the exact set is hard.

Acknowledgments

Thanks to Denise Draper, Ken Fertig, Moises Goldszmidt, Mark Peot and Tom Dean for comments on the paper and discussion of Markov techniques and planning. This research was funded by Rockwell and by ARPA contract F33615-94-C-4426.

References

[Boutilier94] Boutilier, C., and Dearden, R., Using Abstractions for Decision-Theoretic Planning with Time Constraints, in *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1016-1022, 1994.

[Draper94] Draper, D., Hanks, S., and Weld, D., Probabilistic Planning with Information Gathering and Contingent Execution, in *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, AAAI Press, 1994.

[Kushmerick94] Kushmerick, N., Hanks, S., and Weld, D., An Algorithm for Probabilistic Planning, *Artificial Intelligence*, to appear.

[Winston94] Winston, W., *Operations research: Applications and Algorithms*, Third edition, Duxbury Press, 1994.